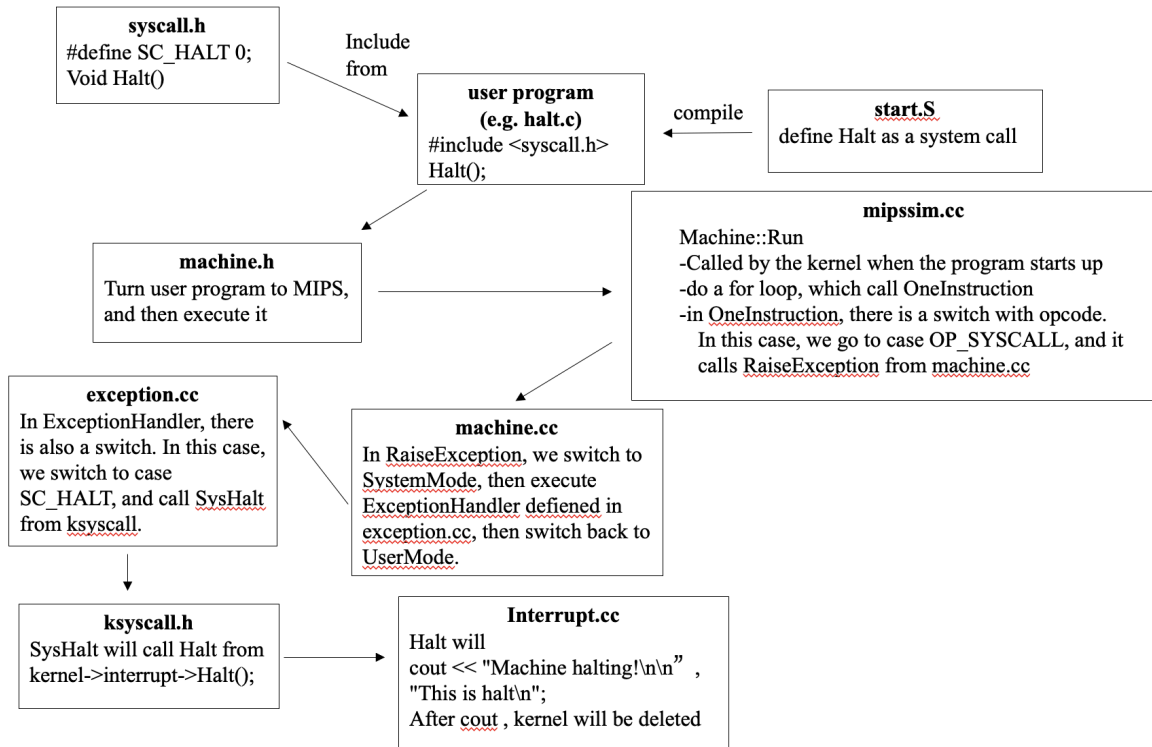


# AFS\_HW01\_GROUP\_35

## Part 1: Trace Code Result

### 1. Flow Chart of Halt() System Call:



### 2. Details of Trace Halt() Code

#### 1. User program(e.g. halt.c)

```
Halt();
```

#### 2. machine/machine.h

```
Run()          // defined in mipssim.cc
for (;;) {
    OneInstruction(instr);
    kernel->interrupt->OneTick();
    if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
```

```

        Debugger();
    }

```

### 3. machine/mipssim.cc

```

case OP_SYSCALL:
    RaiseException(SyscallException, 0);

```

### 4. machine/machine.cc

```

void
Machine::RaiseException(ExceptionType which, int badVAddr)
{
    DEBUG(dbgMach, "Exception: " << exceptionNames[which]);
    registers[BadVAddrReg] = badVAddr;
    DelayedLoad(0, 0);    // finish anything in progress
    kernel->interrupt->setStatus(SystemMode);
    ExceptionHandler(which);    // interrupts are enabled at this point
    kernel->interrupt->setStatus(UserMode);
}

```

```

ExceptionHandler
    switch(type)
        case SC_Halt:
            DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
            SysHalt();
            cout<<"in exception\n";
            ASSERTNOTREACHED();
            break;

```

```

void SysHalt()
{

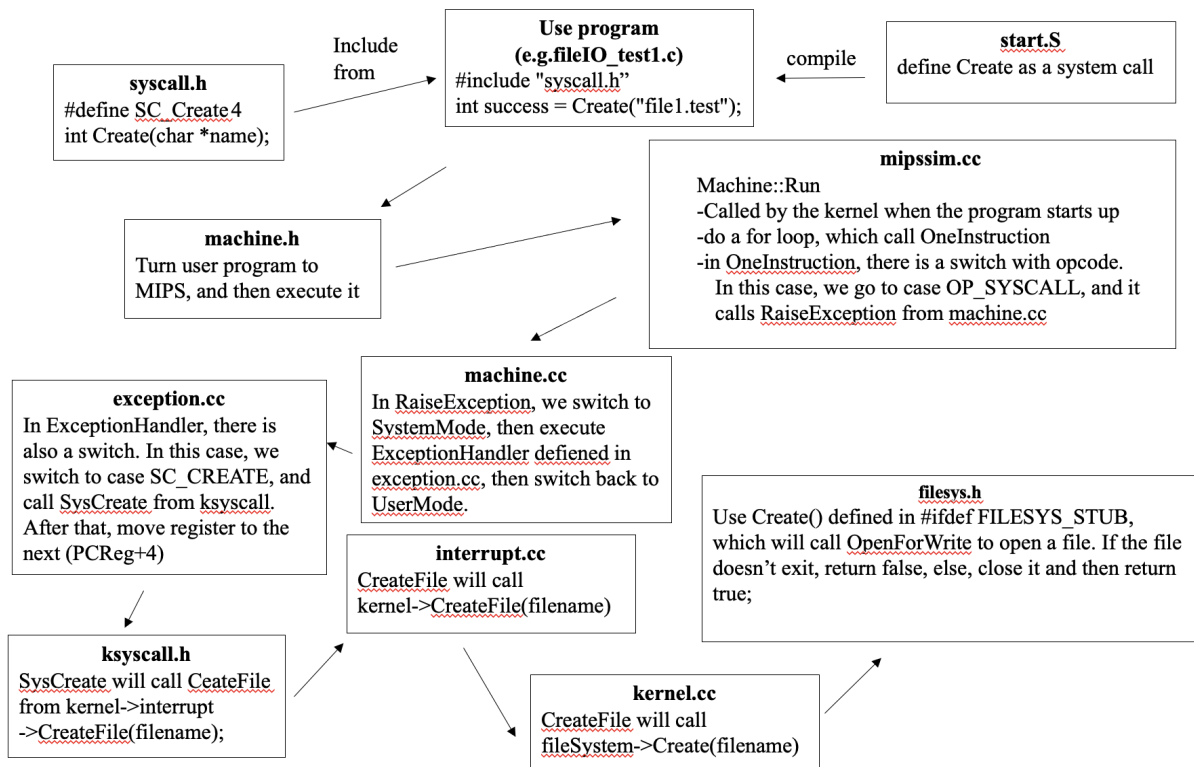
```

```
kernel->interrupt->Halt();  
}
```

5. machine/interrupt.cc

```
void  
Interrupt::Halt()  
{  
    cout << "Machine halting!\n\n";  
    cout << "This is halt\n";  
    kernel->stats->Print();  
    delete kernel; // Never returns.  
}
```

### 3. Flow Chart of Create() System Call:



### 4. Details of Trace Create() Code

#### 1. User program(e.g. fileIO\_test1.c)

```
int success = Create("file1.test");
```

#### 2. machine/machine.h

```
Run()          // defined in mipssim.cc
{
    for (;;) {
        OneInstruction(instr);
        kernel->interrupt->OneTick();
        if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
```

```
        Debugger();  
    }
```

3. machine/mipssim.cc

```
case OP_SYSCALL:  
    RaiseException(SyscallException, 0);
```

4. machine/machine.cc

```
void  
Machine::RaiseException(ExceptionType which, int badVAddr)  
{  
    DEBUG(dbgMach, "Exception: " << exceptionNames[which]);  
    registers[BadVAddrReg] = badVAddr;  
    DelayedLoad(0, 0);    // finish anything in progress  
    kernel->interrupt->setStatus(SystemMode);  
    ExceptionHandler(which);    // interrupts are enabled at this point  
    kernel->interrupt->setStatus(UserMode);  
}
```

```
ExceptionHandler  
    switch(type)  
    case SC_Create:  
        val = kernel->machine->ReadRegister(4);  
        {  
            char *filename = &(kernel->machine->mainMemory[val]);  
            //cout << filename << endl;  
            status = SysCreate(filename);  
            kernel->machine->WriteRegister(2, (int) status);  
        }  
        kernel->machine->WriteRegister(PrevPCReg,
```

```

        kernel->machine ->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg,
        kernel->machine ->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg,
        kernel->machine->ReadRegister(PCReg)+4);
    return;
    ASSERTNOTREACHED();
    break;

```

## 5. filesystems/filesys.h

```

bool Create(char *name) {
    int fileDescriptor = OpenForWrite(name);
    if (fileDescriptor == -1) return FALSE;
    Close(fileDescriptor);
    return TRUE;
}

```

## 5. Details of Makefile

When compiling a program, for example, *halt*.

1. When linking using *ld*, the program (*halt*) will require its object files and *start.o* for syscall library. After that use *coff2noff* to change format.

```

halt: halt.o start.o
    $(LD) $(LDFLAGS) start.o halt.o -o halt.coff
    $(COFF2NOFF) halt.coff halt

```

2. Its object files (*halt.o* and *start.o*) will be compiled using *gcc*. Also, the flags indicate that the binaries should be found in those two directories. *start.S* is needed in order to run *main* function in a C program.

```

CFLAGS = -G 0 -c $(INCDIR)

```

```
-B../usr/local/nachos/lib/gcc-lib/decstation-ultrix/2.95.2/
-B../usr/local/nachos/decstation-ultrix/bin/
start.o: start.S ../userprog/syscall.h
        $(CC) $(CFLAGS) $(ASFLAGS) -c start.S

halt.o: halt.c
        $(CC) $(CFLAGS) -c halt.c
```

### 3. The result.

```
halt
Machine halting! // from Interrupt::Halt()

This is halt // from Interrupt::Halt()
// from Statistics::Print(), called by Interrupt::Halt()
Ticks: total 52, idle 0, system 40, user 12
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
OK (0.013 sec real, 0.013 sec wall)
```

## Part 2: Implement System Call

### 1. Detail of your Console I/O system call implementation

- userprog/syscall.h:

```
#define SC_PrintInt    16

void PrintInt(int number);
```

- test/start.S:

```
.globl PrintInt
.ent    PrintInt
PrintInt:
    addiu    $2,$0,SC_PrintInt
    syscall
```

```
j    $31
.end  PrintInt
```

- userprog/exception.cc:

```
case SC_PrintInt:
    DEBUG(dbgAddr, "Printing int\n");
    val = (int)kernel->machine->ReadRegister(4);
    SysPrintInt(val);

    kernel->machine->WriteRegister(PrevPCReg,
                                   kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg,
                                   kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg,
                                   kernel->machine->ReadRegister(PCReg) + 4);

    return;
    ASSERTNOTREACHED();
    break;
```

- userprog/ksyscall.h:

\* 在interrupt中無法呼叫kernel->synchConsoleOut->PutString

```
void SysPrintInt(int value) {
    static char zero[3] = "0\n";

    if (value == 0) {
        kernel->synchConsoleOut->PutString(zero);
        return;
    }

    char outputBuf[INT_BUF_LENGTH];
    bool isNeg = false;
    int curPos = INT_BUF_LENGTH;
    outputBuf[--curPos] = '\0';
    outputBuf[--curPos] = '\n';

    if (value < 0) {
        isNeg = true;
        value = -value;
    }

    while (value > 0) {
        outputBuf[--curPos] = '0' + (value % 10);
        value /= 10;
    }
}
```



```

    }

    if (isNeg)
        outputBuf[--curPos] = '-';

    kernel->synchConsoleOut->PutString(&outputBuf[curPos]);
}

```

- userprog/synchconsole.h:

\* 因為怕有其他使用PutChar的部分, 因此額外寫一個PutString

```
void PutString(char *ch);
```

- userprog/synchconsole.cc:

```

void
SynchConsoleOutput::PutString(char *str)
{
    lock->Acquire();
    consoleOutput->PutString(str);
    waitFor->P();
    lock->Release();
}

```

- machine/console.h:

```
void PutString(char *str);
```

- machine/console.cc:

```

void
ConsoleOutput::PutString(char *str)
{
    ASSERT(putBusy == FALSE);
    WriteFile(writeFileNo, str, strlen(str));
    putBusy = TRUE;
    kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
}

```

- result:

- consoleIO\_test1:

```
===== consoleIO_test1 =====
```

```
9
8
7
6
consoleIO_test1
Machine halting!

This is halt
Ticks: total 669, idle 400, system 180, user 89
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 4
Paging: faults 0
Network I/O: packets received 0, sent 0
```

- consoleIO\_test2:

```
===== consoleIO_test2 =====
15
16
17
18
19
consoleIO_test2
Machine halting!

This is halt
Ticks: total 826, idle 500, system 220, user 106
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 5
Paging: faults 0
Network I/O: packets received 0, sent 0
```

## 2. Detail of your File I/O system call implementation

- userprog/ksyscall.h:

```
OpenFileId SysOpen(char *name) {
    for (int i = 0; i < 20; i++)
        if (kernel->fileSystem->fileDescriptorTable[i] == NULL) {
            kernel->fileSystem->fileDescriptorTable[i]
                = kernel->fileSystem->Open(name);
            if (kernel->fileSystem->fileDescriptorTable[i] == NULL)
                return -1;
            return i + 2; // OpenFileId starts from 2 (0=stdin, 1=stdout reserved)
        }
    return -1;
}
```

```

}

int SysWrite(char *buffer, int size, OpenFileId id) {
    if (size < 0) // Invalid size
        return -1;
    if (id == 1) { // Console output (stdout)
        for (int i = 0; i < size; i++)
            kernel->synchConsoleOut->PutChar(buffer[i]);
        return size;
    }
    int idx = id - 2; // Convert to 0-based index (ids 2+ are files)
    if (idx < 0 || idx >= 20 || kernel->fileSystem->fileDescriptorTable[idx] == NULL)
        return -1;
    return kernel->fileSystem->fileDescriptorTable[idx]->Write(buffer, size);
}

int SysRead(char *buffer, int size, OpenFileId id) {
    if (size < 0) // Invalid size
        return -1;
    if (id == 0) { // Console input (stdin)
        for (int i = 0; i < size; i++)
            buffer[i] = kernel->synchConsoleIn->GetChar();
        return size;
    }
    int idx = id - 2; // Convert to 0-based index (ids 2+ are files)
    if (idx < 0 || idx >= 20 || kernel->fileSystem->fileDescriptorTable[idx] == NULL)
        return -1;
    return kernel->fileSystem->fileDescriptorTable[idx]->Read(buffer, size);
}

int SysClose(OpenFileId id) {
    if (id == 0 || id == 1) // Cannot close console
        return 0;
    int idx = id - 2; // Convert to 0-based index (ids 2+ are files)
    if (idx < 0 || idx >= 20 || kernel->fileSystem->fileDescriptorTable[idx] == NULL)
        return 0;
    delete kernel->fileSystem->fileDescriptorTable[idx];
    kernel->fileSystem->fileDescriptorTable[idx] = NULL;
    return 1;
}

```

- userprog/exception.cc

```

case SC_Open:
    DEBUG(dbgAddr, "Open file\n");

    {
        val = kernel->machine->ReadRegister(4);
    }

```

```

char *name = &(kernel->machine->mainMemory[val]);
OpenFileId ret = SysOpen(name);
kernel->machine->WriteRegister(2, ret);
}

kernel->machine->WriteRegister(PrevPCReg,
                             kernel->machine->ReadRegister(PCReg));
kernel->machine->WriteRegister(PCReg,
                             kernel->machine->ReadRegister(PCReg) + 4);
kernel->machine->WriteRegister(NextPCReg,
                             kernel->machine->ReadRegister(PCReg) + 4);

return;
ASSERTNOTREACHED();
break;
case SC_Read:
    DEBUG(dbgAddr, "Read file\n");

    {
        val = kernel->machine->ReadRegister(4);
        char *buffer = &(kernel->machine->mainMemory[val]);
        int size = kernel->machine->ReadRegister(5);
        OpenFileId id = (OpenFileId)kernel->machine->ReadRegister(6);
        int ret = SysRead(buffer, size, id);
        kernel->machine->WriteRegister(2, ret);
    }

    kernel->machine->WriteRegister(PrevPCReg,
                             kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg,
                             kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg,
                             kernel->machine->ReadRegister(PCReg) + 4);

    return;
    ASSERTNOTREACHED();
    break;
case SC_Write:
    DEBUG(dbgAddr, "Write file\n");

    {
        val = kernel->machine->ReadRegister(4);
        char *buffer = &(kernel->machine->mainMemory[val]);
        int size = kernel->machine->ReadRegister(5);
        OpenFileId id = (OpenFileId)kernel->machine->ReadRegister(6);
        // fprintf(stderr, "buffer: %p\n", buffer);
        // cerr << "size: " << size << endl;
        // cerr << "id: " << id << endl;
        int ret = SysWrite(buffer, size, id);
    }

```

```

        kernel->machine->WriteRegister(2, ret);
    }

    kernel->machine->WriteRegister(PrevPCReg,
                                   kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg,
                                   kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg,
                                   kernel->machine->ReadRegister(PCReg) + 4);

    return;
    ASSERTNOTREACHED();
    break;
case SC_Close:
    DEBUG(dbgAddr, "Close file\n");

    {
        OpenFileId id = (OpenFileId)kernel->machine->ReadRegister(4);
        int ret = SysClose(id);
        kernel->machine->WriteRegister(2, ret);
    }

    kernel->machine->WriteRegister(PrevPCReg,
                                   kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg,
                                   kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg,
                                   kernel->machine->ReadRegister(PCReg) + 4);

    return;
    ASSERTNOTREACHED();
    break;

```